5.3.4. Lineare Filter:

Betrachte Vektor v, dessen Komponenten wieder diskrete Werte einer Funktion oder eines Bildes darstellen (Sampling).

Zunächst 1-dimensional.
Wir *filtern* diesen Vektor, indem wir jede Komponente ersetzen durch eine gewichtete Kombination der Nachbarkomponenten.

So entspricht die *Maske*:

$\frac{1}{4}[1 \quad 2 \quad 1]$ der Operation $v_j = \frac{v_{j-1} + 2v_j + v_{j+1}}{4}$,

bzw. der Matrixmultiplikation

$$v \to \frac{1}{4}\begin{pmatrix} 2 & 1 & & & \\ 1 & 2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 2 & 1 \\ & & & 1 & 2 \end{pmatrix} \cdot v$$

Vorsicht am Rand! Fehlende Werte haben Einfluß auf Messwerte!
Man benötigt Annahmen für v_0 und v_{n+1}. Was ist sinnvoll?

Im zwei-dim. entspricht dies z.B. der Maske

$$\frac{1}{8}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \rightarrow \quad v_{i,j} = \frac{v_{i,j-1} + v_{i-1,j} + 4v_{i,j} + v_{i,j+1} + v_{i+1,j}}{8},$$

Genauso Gaussfilter:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Dies entspricht glättenden Filtern, die zu einem weicheren Bild führen:
Mittelwertfilter, Glätter, Weichzeichner, oder Tiefpassfilter zur Abschwächung von Rauschen, bzw. hochfrequenten Anteilen.
Hochfrequente Störungen, die einzelne Komponente verändern, werden durch die Mittelwertbildung verringert!

Entsprechend kann man Hochpassfilter definieren, die die Unterschiede hervorheben, das Bild härter machen und niederfrequente (glatte) Anteile abschwächen (Differenzfilter, Scharfzeichner).

z.B. Laplacefilter:
(Sobelfilter)

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

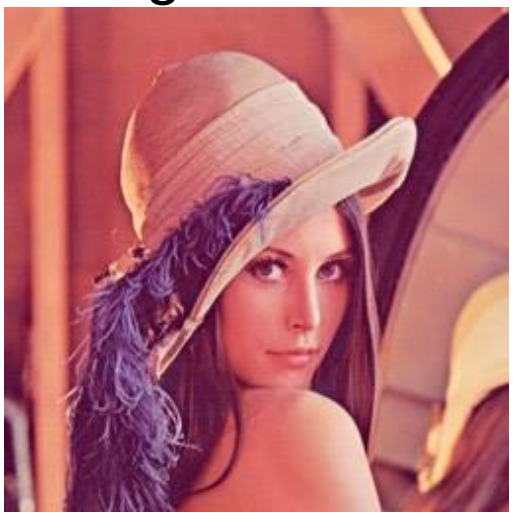
Bisher entsprechen die Methoden einer *Filterung im Ortsraum*.
Im Gegensatz dazu kann man auch *Filter im Phasenraum*
(Frequenz~) zum Entfernen von Rauschen (Noise) verwenden:

v → DFT(v) → Filter → IDFT(v)

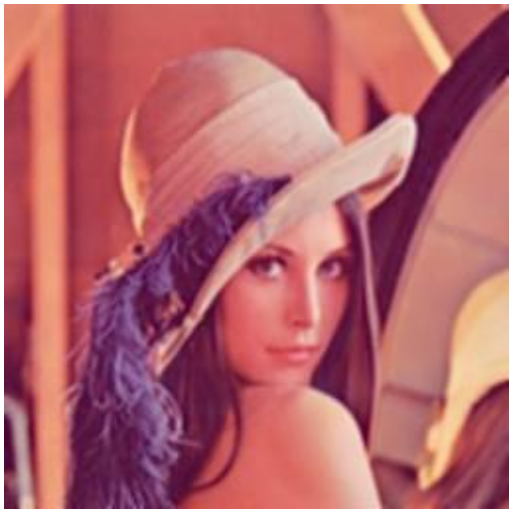
Also Transformation in (Fourier-)Koeffizientenraum,
dann Filtern der Koeffizienten = Gewichten/Löschen,
dann Rücktransformation (vgl. MP3)
Hochpassfilter, Tiefpass, Filterband,…

Filtern eines Bildes:

Original



Mittelwertfilter



Reduktion

Differenzfilter

Reduktion

5.3.5. Wavelets als rekusive Filter:

Betrachte **Kombination** von Tiefpassfilter und Hochpassfilter im 1-Dimensionalen zu Masken

$$\begin{bmatrix}1 & 1\end{bmatrix}/2 \qquad \text{und} \qquad \begin{bmatrix}1 & -1\end{bmatrix}/2$$

Ersetze den ursprünglichen Vektor verlustfrei durch tief-, bzw. hochpass-gefilterte Vektoren halber Länge (down sampling):

$$\begin{pmatrix} v_t \\ \hline v_h \end{pmatrix} = \frac{1}{2} \left(\begin{array}{cccccc} 1 & 1 & & & & \\ & & 1 & 1 & & \\ & & & & \ddots & \ddots \\ \hline 1 & -1 & & & & \\ & & 1 & -1 & & \\ & & & & \ddots & \ddots \end{array} \right) \cdot v$$

oder

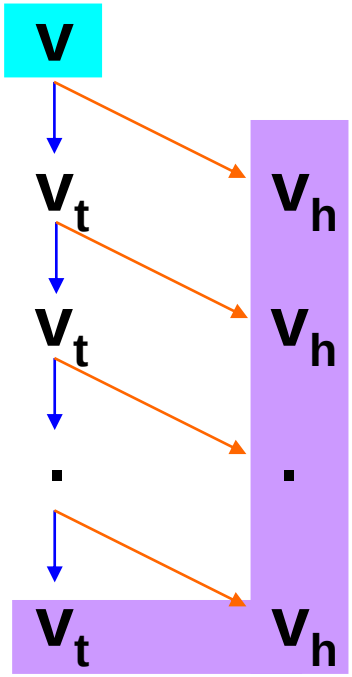
$$\frac{1}{2}\begin{pmatrix} \boxed{\begin{matrix}1 & 1\\ 1 & -1\end{matrix}} & & & & \\ & \boxed{\begin{matrix}1 & 1\\ 1 & -1\end{matrix}} & \ddots & & \\ & & \ddots & \ddots & \\ & & & \ddots & \\ & & & & \boxed{\begin{matrix}1 & 1\\ 1 & -1\end{matrix}} \end{pmatrix} \cdot v$$

(nur anders sortiert)

Der Differenzanteil v_h ist in der Regel klein und wird nicht weiterbearbeitet!

Der Mittelwertanteil (tiefpassgefiltert) v_t wird nach demselben Schema weiteraufgespalten wiederum in Tief/Hochpassanteil durch dieselben Masken.

Insgesamt:



Level 0 , Länge n

Level 1 , n/2

Level 2 , n/4

.

Level p , 1

Ersetze nun den Ausgangsvektor $v \in \mathbf{R}^n$ durch den letzten Mittelwertanteil auf Level p und sämtliche Differenzanteile v_h , das sind auch genau n Zahlen.
(v_h und v_t entsprechen quasi den Fourierkoeffizienten)

Vorteile:

Gesamtkosten der Transformation O(n)! Warum?
Differenz-Anteile meist klein → gut komprimierbar

Differenzanteil auf Level k enthält Information über das Verhalten von v auf diesem Level, bzw. in dieser Auflösung
→

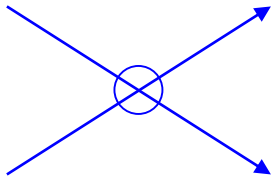
Multiskalenanalyse, Zerlegung des Vektors in verschiedene Frequenzbereiche = Skalen
Entspricht in etwa der Fourier-Analyse.

Filter müssen sparse sein damit billig,
leicht umkehrbar (invertierbar) damit Original rekonstruierbar!
(vgl. DFT und IDFT)
Aufspalten in Frequenzanteil (hoch – tief).

Algorithmische Ähnlichkeit zur Fourier-Transformation am Beispiel des Haar-Filters mit den Masken

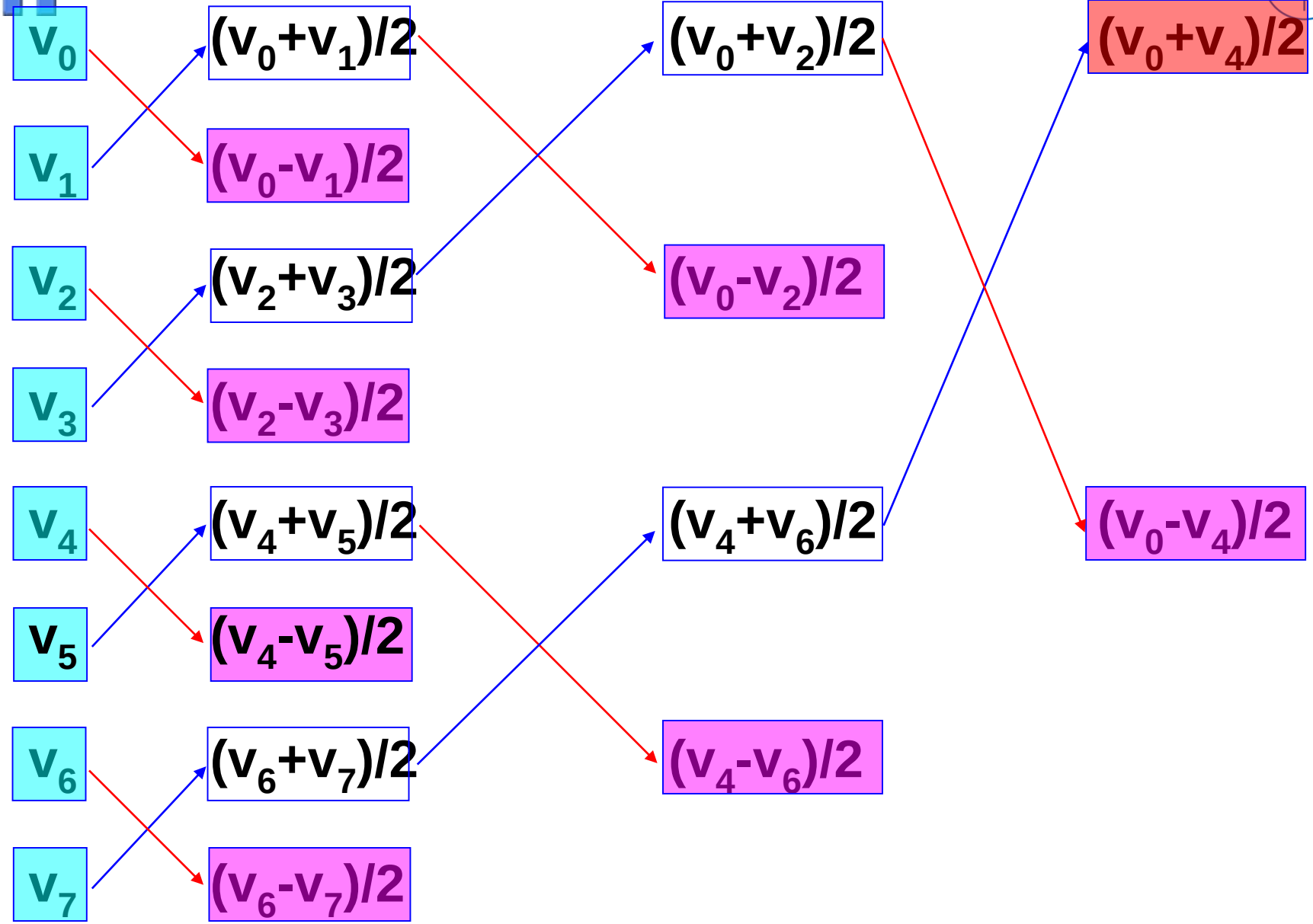
$$\frac{1}{2}[1 \quad 1] \qquad \text{und} \qquad \frac{1}{2}[1 \quad -1] \;:$$

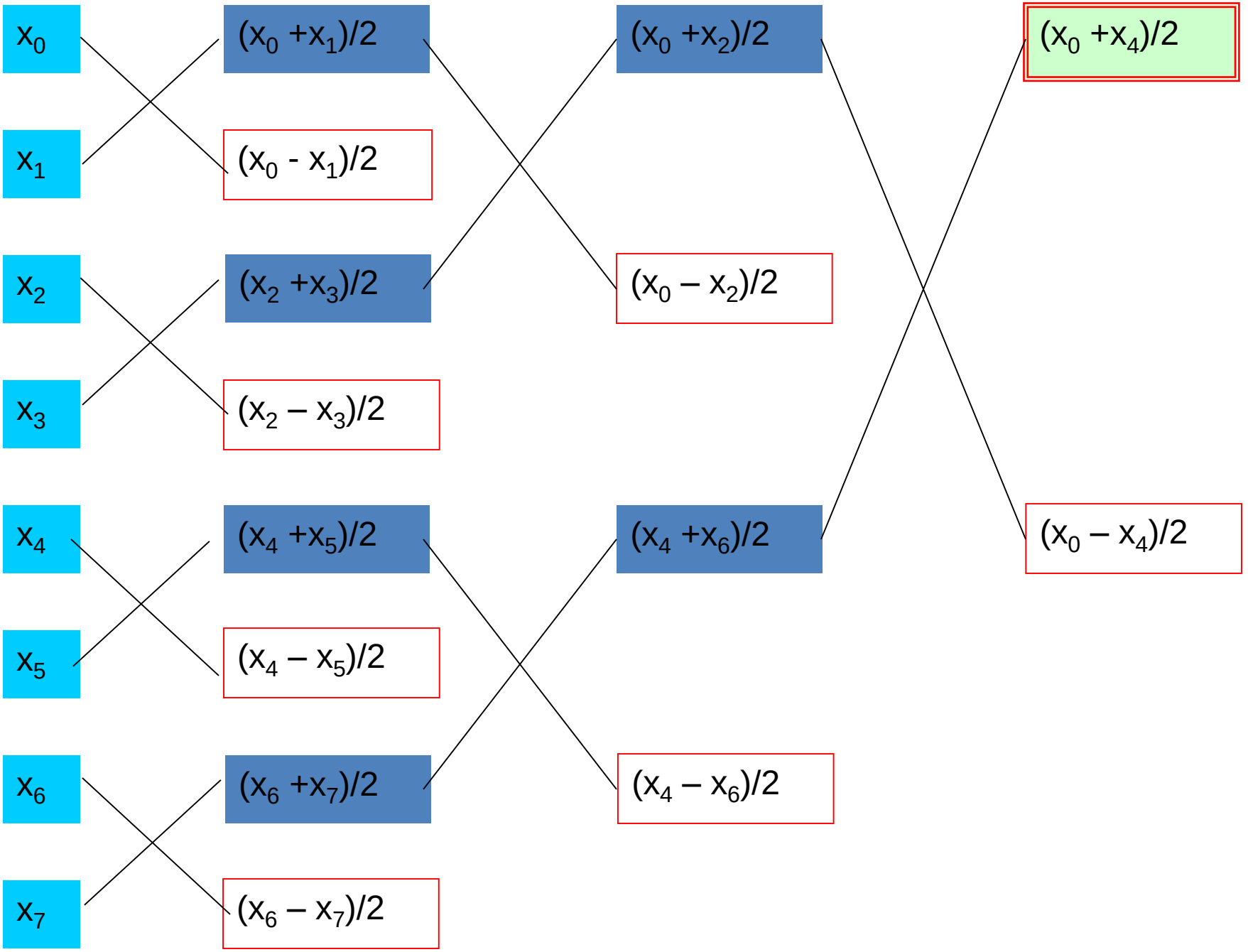
Butterfly



u → **(u+v)/2**

v → **(u-v)/2**


v_0
v_1
v_2
v_3
v_4
v_5
v_6
v_7
$(v_0+v_1)/2$
$(v_0-v_1)/2$
$(v_2+v_3)/2$
$(v_2-v_3)/2$
$(v_4+v_5)/2$
$(v_4-v_5)/2$
$(v_6+v_7)/2$
$(v_6-v_7)/2$
$(v_0+v_2)/2$
$(v_0-v_2)/2$
$(v_4+v_6)/2$
$(v_4-v_6)/2$
$(v_0+v_4)/2$
$(v_0-v_4)/2$


x_0
x_1
x_2
x_3
x_4
x_5
x_6
x_7
$(x_0 + x_1)/2$
$(x_0 - x_1)/2$
$(x_2 + x_3)/2$
$(x_2 - x_3)/2$
$(x_4 + x_5)/2$
$(x_4 - x_5)/2$
$(x_6 + x_7)/2$
$(x_6 - x_7)/2$
$(x_0 + x_2)/2$
$(x_0 - x_2)/2$
$(x_4 + x_6)/2$
$(x_4 - x_6)/2$
$(x_0 + x_4)/2$
$(x_0 - x_4)/2$

v wird dabei transformiert in
alle Differenzanteile und
den letzten Mittelwert.

Unterschied zur DFT:

keine Sortierphase,
einfacherer Butterfly,
nur die Mittelwertanteile werden weiter bearbeitet

→ *O(n)*

Bessere Filterkombination, z.B. Daubechy-Wavelets

$$\left[1+\sqrt{3} \quad 3+\sqrt{3} \quad 3-\sqrt{3} \quad 1-\sqrt{3}\right] / 4\sqrt{2}$$ Mittelwert-Filter

$$\left[1-\sqrt{3} \quad -3+\sqrt{3} \quad 3+\sqrt{3} \quad -1-\sqrt{3}\right] / 4\sqrt{2}$$ Diffferenz-filter

Filterkoeffizienten beschreiben rekursiv eine Funktion.

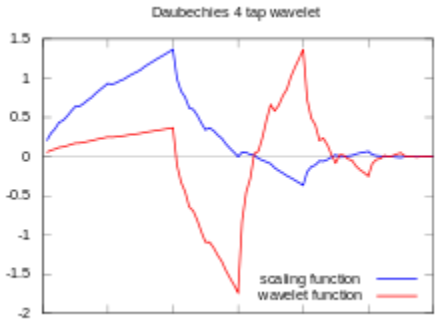
Sie beschreiben den Zusammenhang der Waveletfunktion von einer Skalierung zu einer anderen:

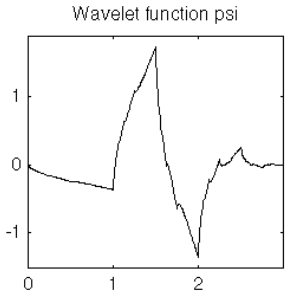
$$\Phi(x) = \sum_k a_k \Phi(2x - k)$$

$$\Phi(2^j x - k), \qquad W(2^j x - k)$$

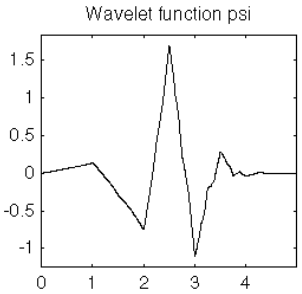
Funktion W(x) erzeugt Basis, vergleiche cos(x).

Daubechies Wavelets (orthogonal)

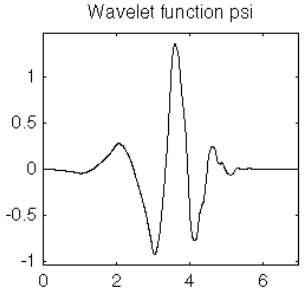




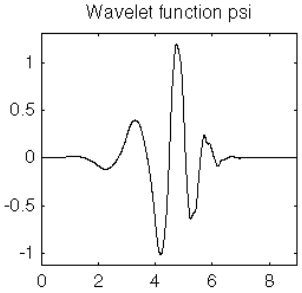
db2



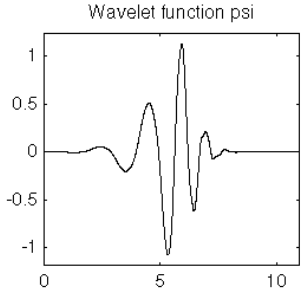
db3



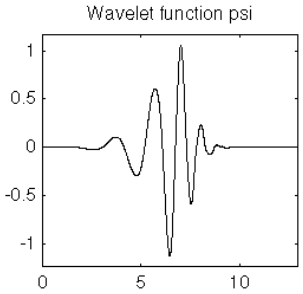
db4



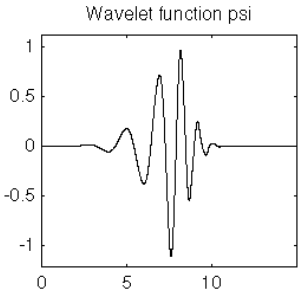
db5



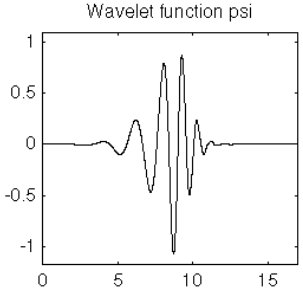
db6



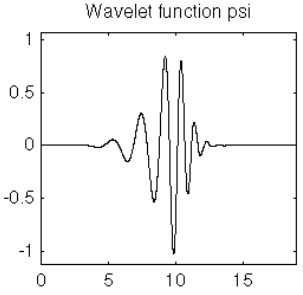
db7



db8

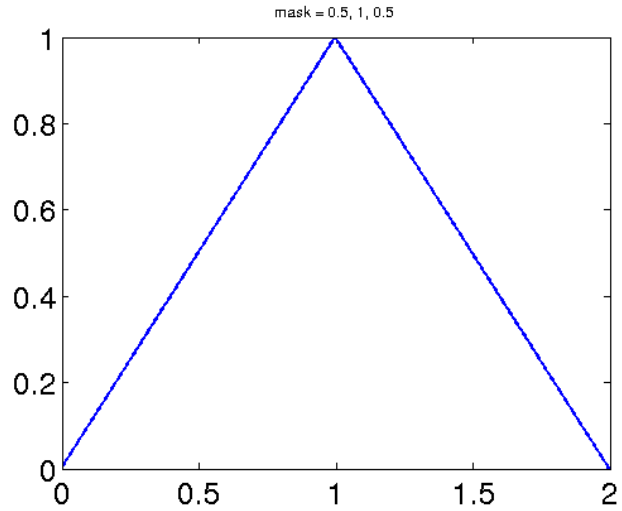


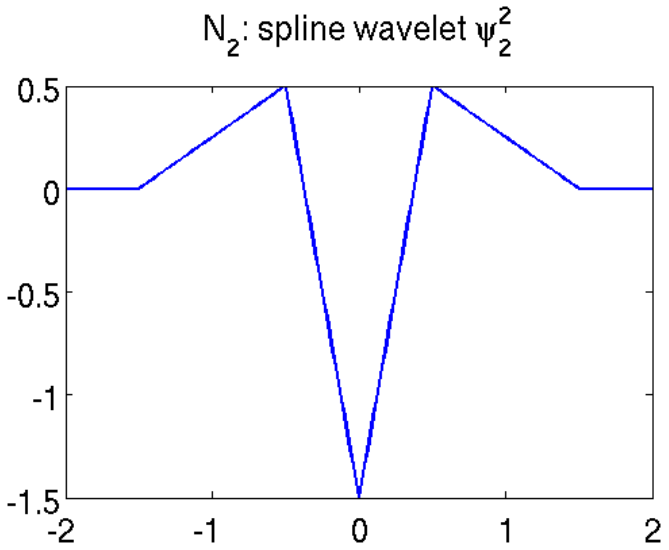
db9

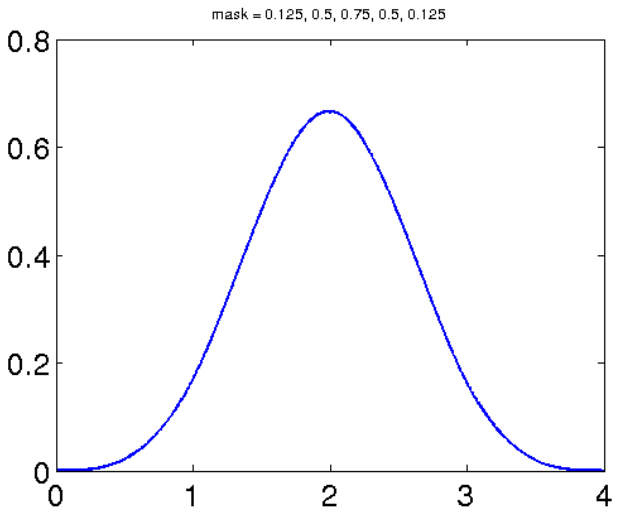


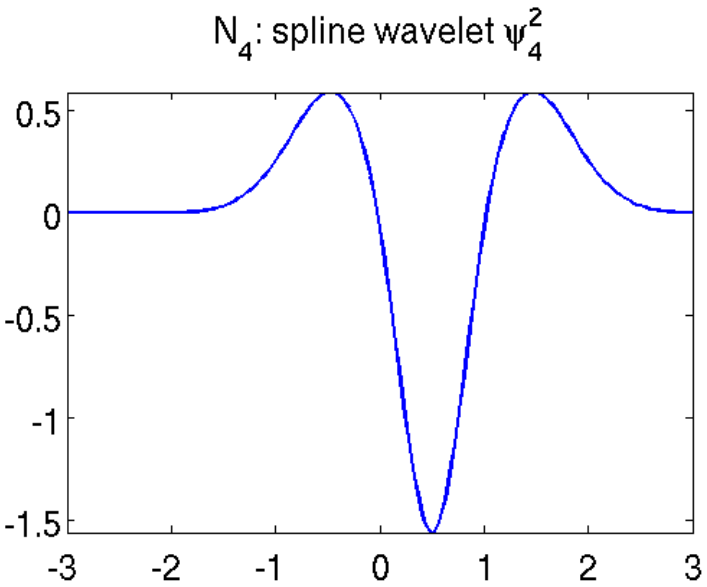
db10

Spline Wavelets

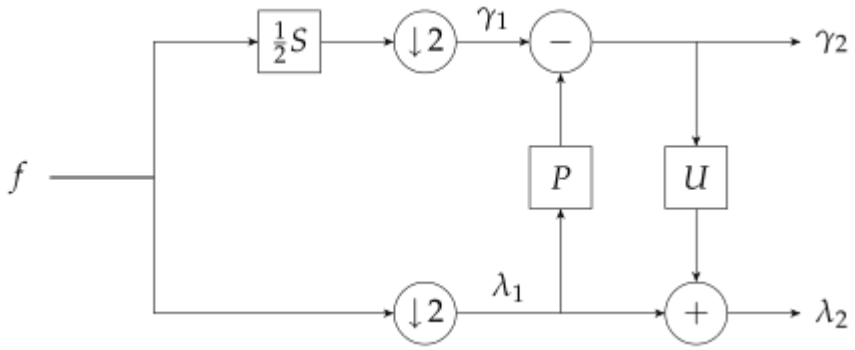




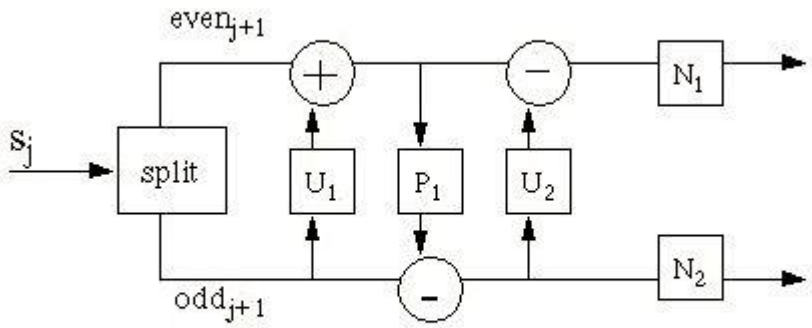




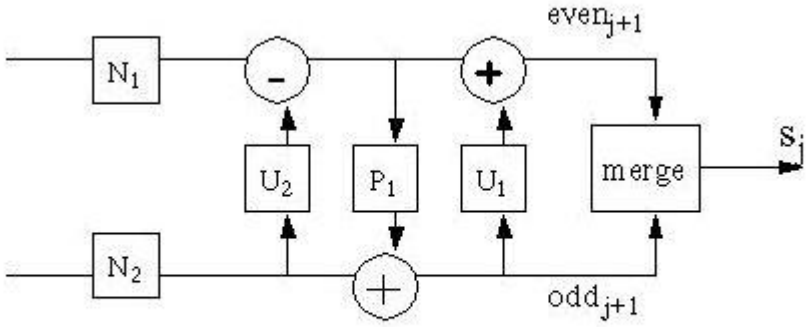
Lifting Scheme (ohne Funktion)



Splitten in Gerade/Ungerade wie Reißverschluss.
Dann zwei oder mehr Filteroperationen wie Predictor/Update.
Vorteil: Leicht umkehrbar.



Daubechies D4 forward wavelet transform

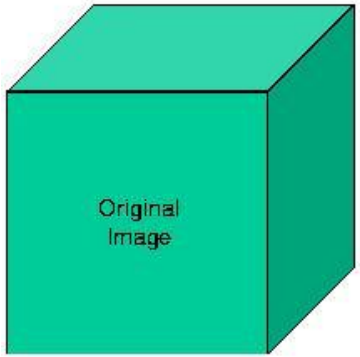


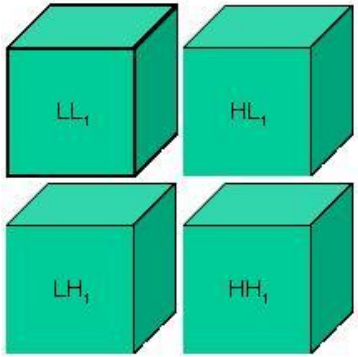
Daubechies D4 inverse wavelet transform

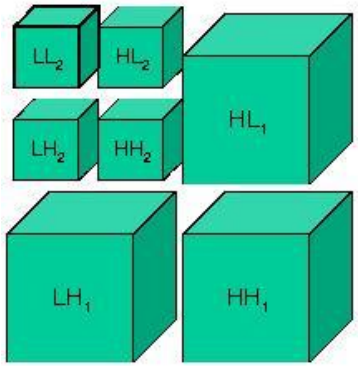
Wavelet als Verschmelzung der
klassischen Filter und
Fourieranalyse
mit der Idee
lokaler Basisfunktionen wie bei den B-Splines.

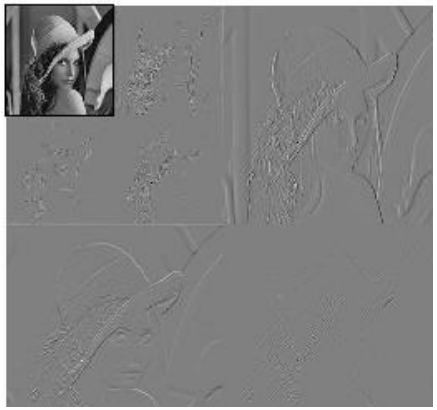
Haar-Filter ←→ Haar-Wavelet

- Orthogonale Wavelet,
- biorthogonale Wavelets,
- Frames,
- Lifting Scheme, …


Original
Image


LL1
HL1
LH1
HH1


LL2
HL2
LH2
HH2
HL1
LH1
HH1



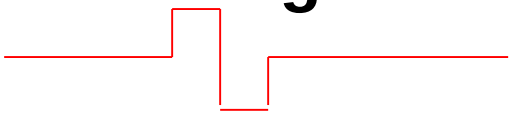
Fourier-Analyse
FFT

cos(kx), sin(kx)
Skalierung k
Alle gleichberechtigt
globaler Träger

keine Variations-
möglichkeit
Kantenproblem
Volle Rekursion
Kosten O(n log(n))
Frequenz aus
Fourierkoeffizienten

Multiskalen-Analyse
Wavelet

$W(2^j x-m)$, $\Phi(2^j x-m)$
Shift m, Skalierung 2^j
Mittelwert/Differenz
lokaler Träger

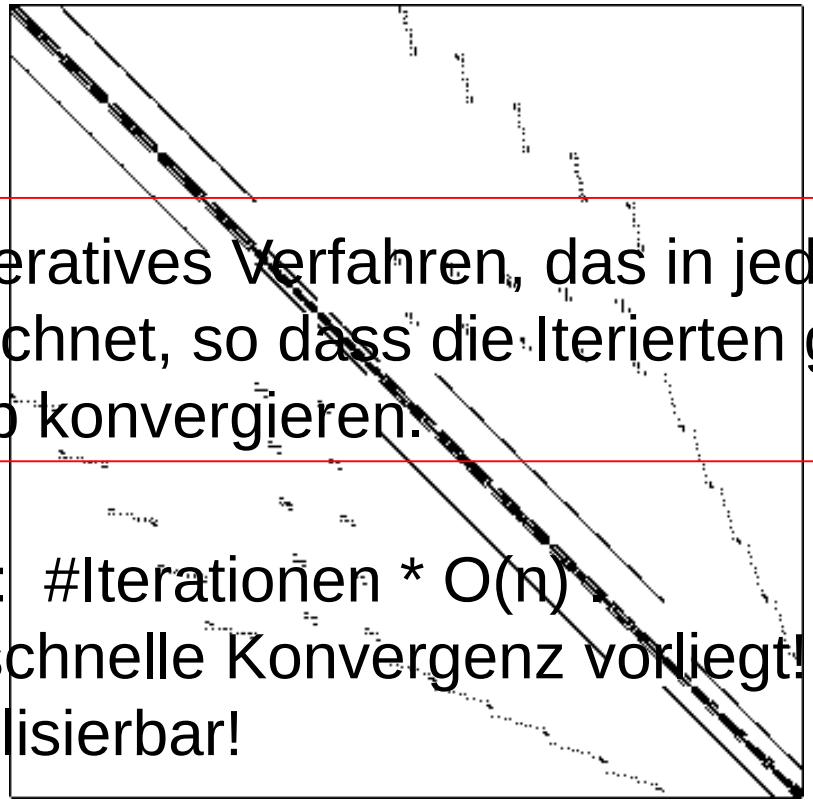


Wahl von Approximations-
güte und Diff'barkeit
Lokal, adaptiv
Rekursion nur im Mittelwert
Kosten O(n)
Frequenz aus
Waveletkoeffizienten

6.3. Iterative Lösung linearer Gleichungssysteme

Großes lineares dünnbesetztes Gleichungssystem $A x = b$

Gauss-Elimination nutzt in der Regel die Dünnbesetztheit nicht aus und führt meist auf Kosten $O(n^3)$;
Im Gegensatz dazu ist oft nur $O(n)$ Speicherbedarf für Matrix A



Idee: Formuliere iteratives Verfahren, das in jedem Schritt nur Matrix*Vektor berechnet, so dass die Iterierten gegen die Lösung von $Ax = b$ konvergieren.

Gesamtkosten: #Iterationen * $O(n)$!
Effizient, falls schnelle Konvergenz vorliegt!
Einfach parallelisierbar!

6.3.1. Stationäre Methoden:

Richardson-Verfahren:
Formulierung eines passenden Fixpunktproblems

$$b = Ax = (A - I)x + x \Leftrightarrow x = b + (I - A)x =: \Phi(x)$$

Daraus ergibt sich die Iteration:

$$x_{k+1} = \Phi(x_k) = b + (I - A)x_k = x_k + (b - Ax_k) = x_k + r_k$$

mit Residuum r_k .

Frage:
Wann konvergiert die ausgehend von einem x_0 definierte Folge x_k gegen die gesuchte Lösung $\bar{x} = A^{-1}b$?

Betrachte dazu den Fehler im k-ten Schritt:

$$\bar{x} - x_{k+1} = \bar{x} - x_k - (b - Ax_k) = \bar{x} - x_k - (A\bar{x} - Ax_k) = (I - A)(\bar{x} - x_k)$$

Ergibt in der Norm $\left\|\bar{x} - x_{k+1}\right\|_2 = \left\|(I - A)(\bar{x} - x_k)\right\|_2 \le \left\|I - A\right\|_2 \cdot \left\|\bar{x} - x_k\right\|_2$

und daher $\left\|\bar{x} - x_k\right\|_2 \le \left\|I - A\right\|_2^k \cdot \left\|\bar{x} - x_0\right\|_2$

Daher liegt Konvergenz vor für $\| I - A \|_2 < 1$.

Dies entspricht der Kontraktionsbedingung für die Iterationsfunktion $\Phi(x)$:

$$\left\|\Phi(x) - \Phi(y)\right\|_2 \le \left\|I - A\right\|_2 \cdot \left\|x - y\right\|_2$$

Richardsonverfahren ist daher nur sinnvoll, wenn $A \approx I$!

Geschickter:

Wende das Verfahren auf das modifizierte Problem an:

$$D = diag(A); \qquad \boxed{\left(D^{-1}A\right)x = \left(D^{-1}b\right)} \quad \Leftrightarrow \quad Ax = b$$

Die Bedingung $D^{-1}A \approx I$ ist besser erfüllt!

Bezeichnung: $\tilde{A} = D^{-1}A$ und $\tilde{b} = D^{-1}b$

ergibt neues Gleichungssystem $\tilde{A}x = \tilde{b}$

Richardson für das tilde-System liefert dann:

$$x_{k+1} = x_k + (\tilde{b} - \tilde{A}x_k) = x_k + D^{-1}(b - Ax_k) \quad \text{oder}$$

$$Dx_{k+1} = Dx_k + (b - Ax_k) = b + (D - A)x_k$$

Dies ist das Jacobiverfahren zur iterativen Lösung von Ax = b

Wesentliche Kosten in jedem Schritt: Ax_k

Konvergent, falls $\left\| I - D^{-1}A \right\|_2 < 1$

Notwendig: Diagonalmatrix D regulär!

Allgemeinere Idee zur Formulierung einer Iterationsfunktion:

Matrix-splitting

$$L: \quad Unterdiagonalteil \quad von \quad A$$

$$U: \quad Oberdiagonalteil \quad von \quad A$$

$$b = Ax = (L + D + U)x = (L + D)x + Ux$$

führt auf Iteration

$$(L + D)x_{k+1} = b - Ux_k = b - (A - L - D)x_k$$

$$= (L + D)x_k + (b - Ax_k)$$

Dies ist das Gauss-Seidel-Verfahren

$$x_{k+1} = x_k + (L+D)^{-1}(b - Ax_k)$$
$$= (L+D)^{-1}b + \left(I - (L+D)^{-1}A\right)x_k$$
$$= (L+D)^{-1}b - (L+D)^{-1}Ux$$

In jedem Schritt ist dabei nur ein Dreiecksgleichungssystem zu lösen.

Das Verfahren ist konvergent, falls $\left\| I - (L+D)^{-1}A \right\|_2 < 1$

Gauss-Seidel ist äquivalent zu Richardson angewendet auf

$$\left((L+D)^{-1}A\right)x = \left((L+D)^{-1}b\right) \quad \Leftrightarrow \quad Ax = b$$

Gauss-Seidel-Iteration

Versuche immer, die neueste Information zu verwenden!

Jacobi Iteration:

$$a_{jj} x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{jm} x_m^{(k)} - \sum_{m=j+1}^{n} a_{jm} x_m^{(k)}$$

Gauss-Seidel Iteration: bereits berechnet

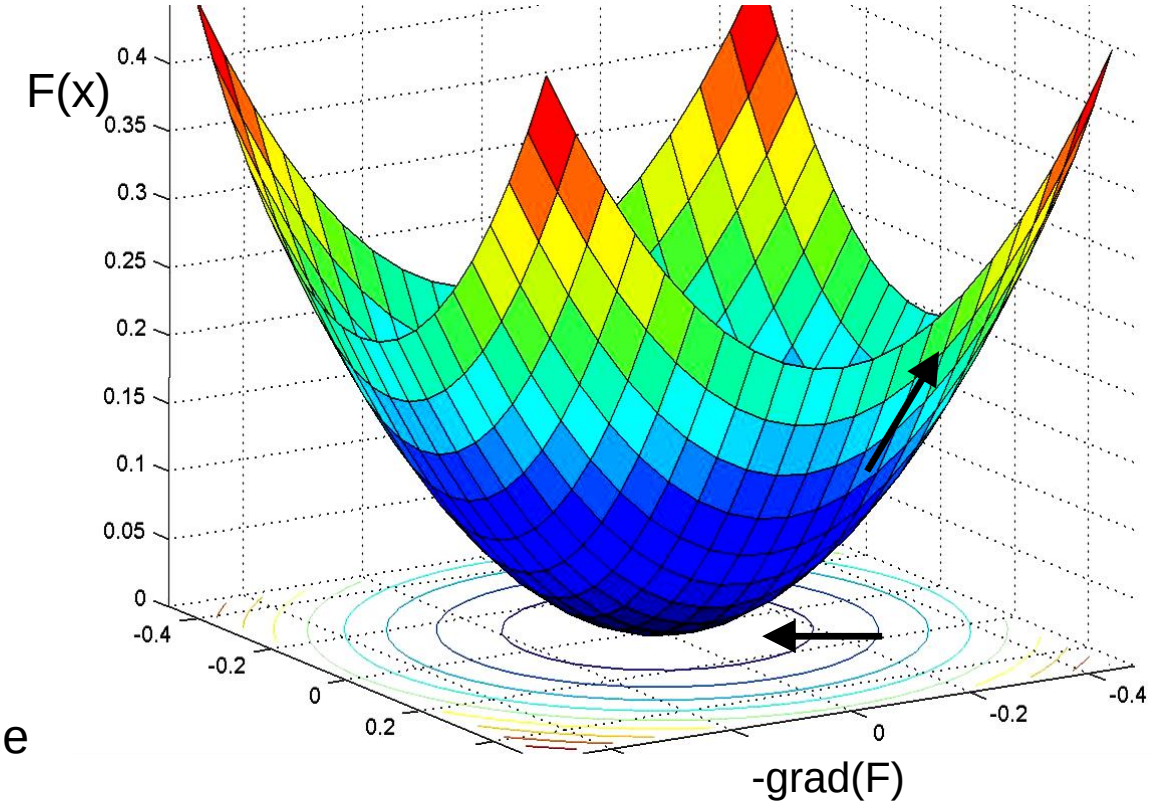
$$a_{jj} x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{jm} x_m^{(k+1)} - \sum_{m=j+1}^{n} a_{jm} x_m^{(k)}$$

6.3.2. Nichtstationäre Verfahren

Das Gradientenverfahren für symmetrisch positiv definite Matrix A:

Betrachte Funktion F(x): $R^n \rightarrow R$, $$F(x) = \frac{1}{2} x^T A x - b^T x$$

F beschreibt einen Paraboloid im R^n:



Minimum dieser Funktion ist wieder der Punkt mit waagrechter Tangente, also die Stelle mit Gradient gleich Null:

$$\nabla F(x) = Ax - b = 0 \Leftrightarrow Ax = b$$

Stelle x, an der der Paraboloid sein Minimum annimmt ist gleich der gesuchten Lösung des Gleichungssystems!

Betrachte Minimierungsaufgabe!

Von aktueller Stelle x_k aus soll die nächste Iterierte x_{k+1} so gewählt werden, dass sie näher am Minimum liegt.

$$x_{k+1} = x_k + \alpha_k d_k$$

mit Suchrichtung d_k und Schrittweite α_k.

Finde Suchrichtung so, dass Funktionswerte kleiner werden:

Abstiegsrichung ist gegeben durch Richtung des negativen Gradienten!

Denn Richtungsableitung in Richtung n ist gleich $\nabla F \cdot n$, und wird am betragsgrößten für $n = \nabla F$ nämlich $\left\|\nabla F\right\|^2$

Daher ist $n = -\nabla F$ lokal die Richtung des steilsten Abstiegs zum Minimum.

Daher verkleinern sich die Funktionswerte auf jeden Fall, wenn man in diese Abstiegsrichtung geht.

Also wähle $$x_{k+1} = x_k - \alpha_k \nabla F(x_k) = x_k + \alpha_k (b - A x_k)$$

Noch zu bestimmten ist optimale Schrittweite α_k, die am nähesten ans Minimum führt!

Betrachte dazu ein-dimensionale Minimierung:

$$\min_\alpha g(\alpha) := \min_\alpha \left(F(x_k + \alpha(b - Ax_k))\right) =$$

$$\min_\alpha \left(\frac{1}{2}(x_k + \alpha d_k)^T A(x_k + \alpha d_k) - b^T (x_k + \alpha d_k) \right) =$$

$$\min_\alpha \left(\frac{1}{2}\alpha^2 d_k^T A d_k - \alpha d_k^T d_k + \frac{1}{2} x_k^T A x_k - x_k^T b \right)$$

mit Lösung $\alpha_k = d_k^T d_k / d_k^T A d_k$, $d_k := b - Ax_k$

Insgesamt:

$$x_{k+1} = x_k + \frac{\|b - Ax_k\|_2^2}{(b - Ax_k)^T A(b - Ax_k)} \cdot (b - Ax_k)$$

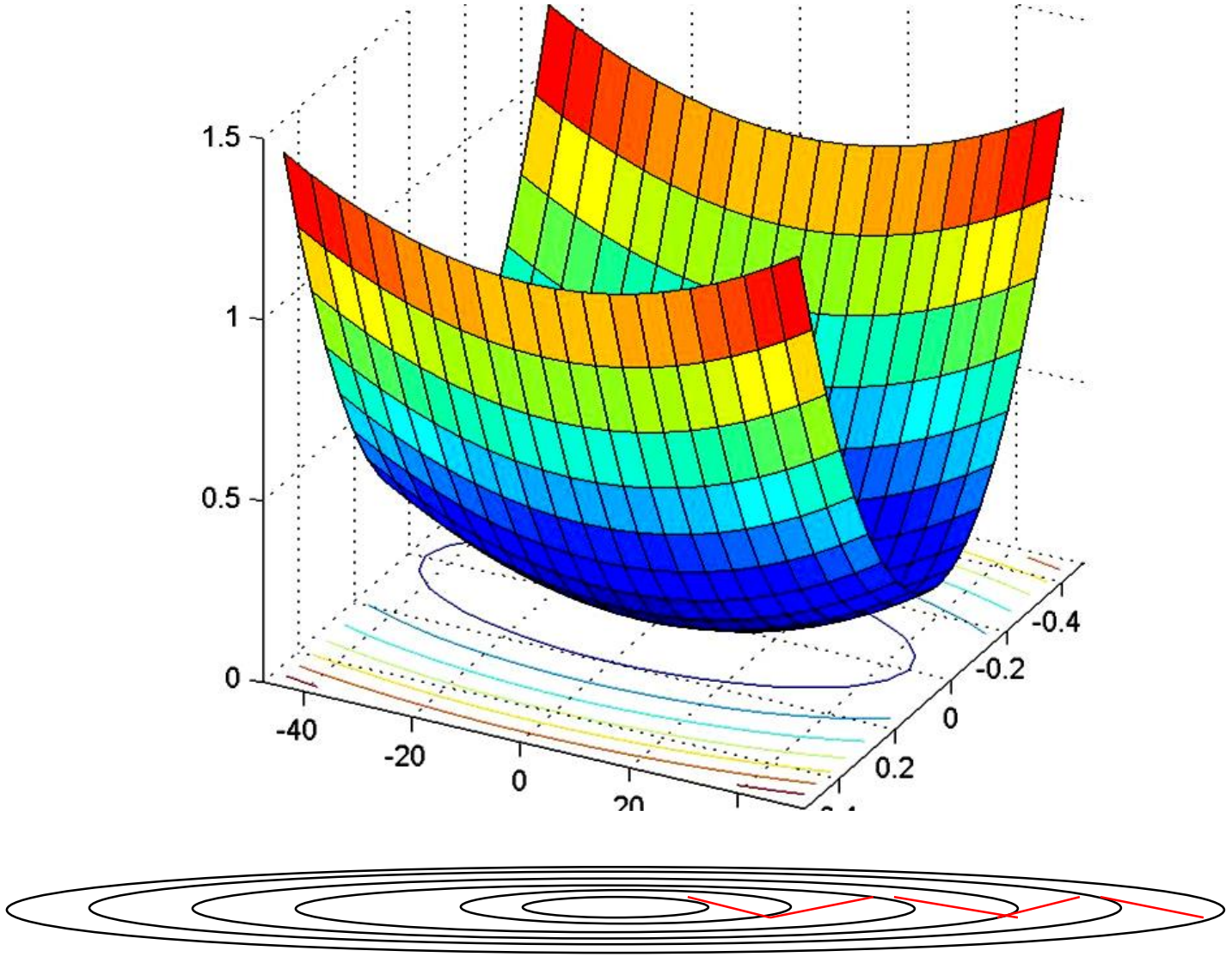
Dazugehörige Fixpunktgleichung ist

$$x = \Phi(x) := x + \frac{\left\| b - Ax \right\|_2^2}{(b - Ax)^T A(b - Ax)} \cdot (b - Ax)$$

mit Fixpunkt $\bar{x} = A^{-1}b$

Ergibt Verfahren des steilsten Abstiegs (‚steepest descent') oder Gradientenverfahren

Nachteil:
Bei stark verzerrtem Paraboloiden ergibt sich sehr langsame Konvergenz.



Für A ≈ I ist der Paraboloid unverzerrt, die Höhenlinien fast kreisförmig → schnelle Konvergenz!

Daher versucht man, Ax=b zu präkonditionieren:

Ersetze Ax=b durch $M^{-1}Ax = M^{-1}b$ mit $M^{-1}A \approx I$

Bessere Variante eines Gradientenverfahren:

Verfahren der konjugierten Gradienten, kurz cg-Verfahren.

Suchrichtung nicht der negative Gradient selbst, sondern die Projektion des Gradienten, so dass alle Suchrichtungen in gewisser Weise orthogonal zueinander sind

Genauer: Suchrichtungen seien A-konjugiert, d.h.

$$d_k^T A d_j = 0 \qquad \text{für} \qquad j \neq k$$

Damit ergibt sich iteratives Verfahren, das nach k Schritten jeweils die beste Näherung an die Lösung in einem k-dimensionalen Unterraum liefert, und daher nach n Schritten fertig ist (in exakter Arithmetik).

Conjugate Gradient Method

Bessere Abstiegsrichtung, global optimal:

$x_{k+1} := x_k + \alpha_k p_k$ als neue Suchrichtung an Stelle des Gradienten

Projektion des Gradienten,
so dass A-conjugate (orthogonal) zu allen
vorherigen Suchrichtungen:

$p_k \perp Ap_j$ for all $j<k$ oder $p_k \perp_A p_j$ oder $p_k^T A p_j = 0$ für $j<k$

α_k wieder durch 1-dimensionale Minimierung wie vorher
(für verwendetes p_k)

Conjugate Gradient Algorithm

$x_0=0$; $r_0 = b - A\,x_0$;

For k=1,2,…: $\beta_{k-1} = r_{k-1}^T r_{k-1} / r_{k-2}^T r_{k-2}$; ($\beta_0=0$)

$p_k = r_{k-1} + \beta_{k-1}\, p_{k-1}$;

$\alpha_k = r_{k-1}^T r_{k-1} / p_k^T A p_k$;

$x_k = x_{k-1} + \alpha\, p_k$;

$r_k = r_{k-1} - \alpha_k\, A p_k$;

if $|| r_k || < \varepsilon$: STOP

Eigenschaften des cg-Verfahrens:

Bestimme in Unterraum $(b, Ab, A^2b, \ldots, A^kb)$ die beste Lösung mit minimalem Fehler: Optimal! Daher nach n Schritten fertig!?

Algorithmus ist billig – in jedem Schritt nur Ax.

In dieser Form nur möglich für symmetrisch positiv definites A.
Für allgemeine Matrix:

- GMRES (optimal, aber teurer)
- BiCG (nicht optimal, aber billig)

Wichtig: Präkonditionierung P*Ax = P*b, Jacobi, Gauss-Seidl

6.5.3. Gradientenverfahren und Neuronale Netze:

Einfaches Neuronales Netz besteht aus n Knoten N_1 , ..., N_n ; jeder Knoten N_i empfängt ein Eingangssignal x_i , verstärkt dieses Signal mit einem Faktor w_i und gibt es an einen gemeinsamen Ausgangsknoten weiter, der alle eingehenden Signale aufsummiert zu

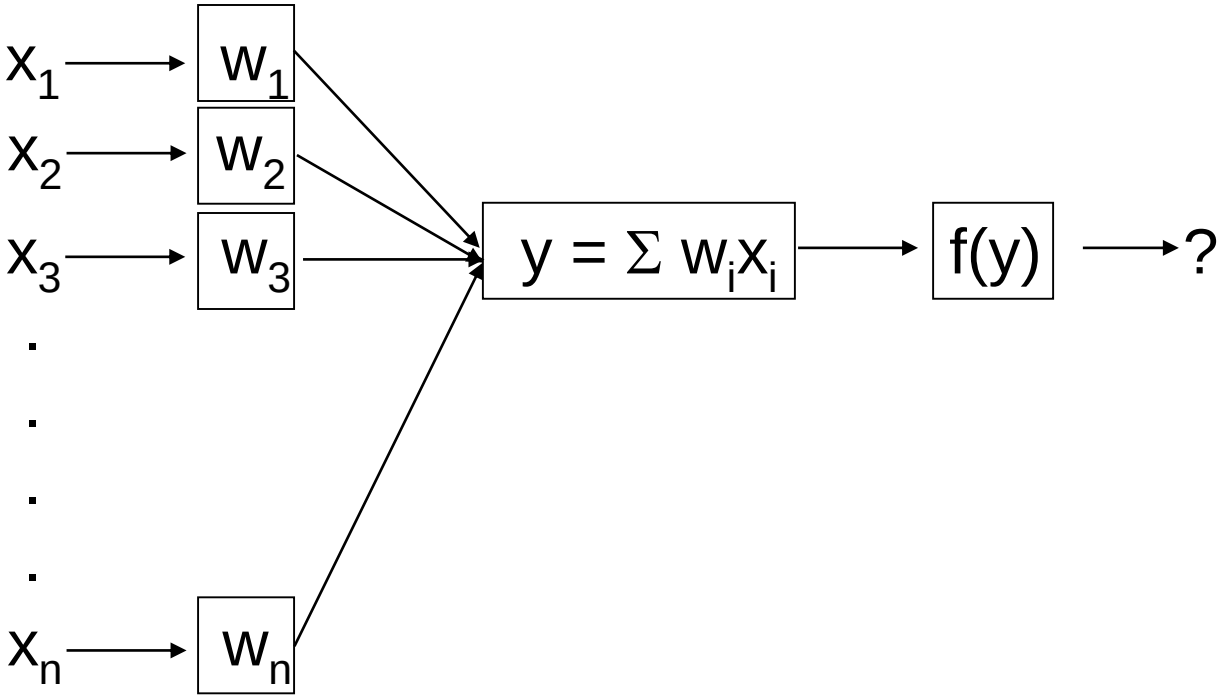
$$y = \sum_{i=1}^{n} w_i x_i$$

Danach kann der Wert y noch weiterbearbeitet werden durch eine Funktion f(.).
Abhängig von dem sich ergebenden Wert von f(y) sollen z.B. Ja/Nein-Entscheidungen getroffen werden:

$$f(y) = \begin{cases} 1 & falls \quad y \geq \alpha \\ 0 & falls \quad y < \alpha \end{cases}$$

entspricht einer Ja/Nein-Antwort

Einschichtiges Neuronales Netz:



Wir wollen mit dem NN einen Vorgang modellieren, bei dem für beliebig viele Beispiele B_j mit Eingabedaten x_j bekannt ist, welcher Ausgangswert y_j zu erwarten ist.

$$w = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}; \qquad y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}; \quad x^{(j)} = \begin{pmatrix} x_1^{(j)} \\ \vdots \\ x_n^{(j)} \end{pmatrix};$$

Wähle nun die Gewichte w so, dass

$$\sum_{i=1}^{n} w_i x_i^{(j)} = w^T x^{(j)} = x^{(j)^T} w \approx y_j, \qquad \textit{für} \quad j = 1,\ldots,m.$$

Also versuche, das NN durch Wahl der Gewichte so zu modellieren, dass es zu den gewünschten Ausgabewerten führt. Dies entspricht einem linearen Ausgleichsproblem der Form

$$\min_w \|Xw - y\| \qquad \textit{mit} \quad X = \begin{pmatrix} x^{(1)^T} & \cdots & x^{(m)^T} \end{pmatrix}$$

Lösung mit Normalgleichung $X^T X w = X^T y$ oder
QR-Zerlegung $X = QR$

Andere Variante:

Schrittweise iterative Verbesserung der Gewichte bei neuen Beispieldaten:

x_{neu} mit gewünschter Ausgabe y_{neu} soll möglichst gut erreicht werden.

Annahme: Bisherige Beispiele haben zu Gewichten w geführt. Einarbeiten der neuen Beispieldaten. Also gesucht:

Nullstelle oder Minimum von $\left(y_{neu} - w^T x_{neu}\right)^2 \stackrel{!}{=} 0$

Gradient dieser Funktion in Abhängigkeit von w:

$$\nabla\left(y_{neu}^2 + (w^T x_{neu})^2 - 2 y_{neu}(w^T x_{neu})\right) = const * x_{neu}$$

ist Suchrichtung zur Verbesserung der aktuellen Gewichte w:

$$w_{neu} = w_{alt} + \alpha x_{neu}$$

liefert

$$y = w_{neu}^T x_{neu} = w_{alt}{}^T x_{neu} + \alpha x_{neu}^T x_{neu} = y_{alt} + \alpha \left\| x_{neu} \right\|^2 \overset{!}{=} y_{neu}$$

Daher sollte $\alpha = \dfrac{y_{neu} - y_{alt}}{\left\| x_{neu} \right\|^2}$ gewählt werden.

Da wir aber nur eine kleine Änderung in Richtung des gewünschten Resultats erreichen wollen, und die bisherigen Testbeispiele, die zu dem alten w geführt haben, nicht vollständig vernachlässigen wollen, definieren wir eine Lernrate $\eta < 1$ und setzen

$$w_{neu} = w_{alt} + \eta \frac{y_{neu} - w_{alt}^T x_{neu}}{\left\| x_{neu} \right\|^2} x_{neu}$$

Lernregel für NN, verwandt mit Gradientenverfahren!